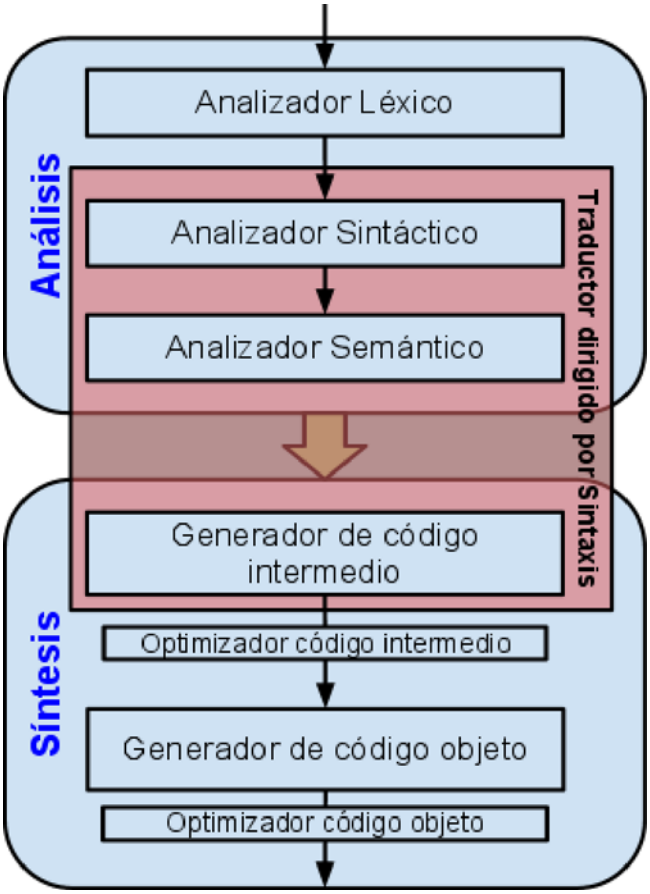
Traductores del lenguaje

Pau Arlandis, Andrés Orcajo, Guillermo Ramos, Álvaro J. Aragoneses.

Introducción



Se realiza la traducción en dos pasos, por dos razones:

- Portabilidad → El código intermedio puede ser único para muchos análisis. Además es independiente de la máquina.
- Sencillez → Es más fácil dar dos pequeños saltos que uno grande.

Podríamos decir que en procesadores del lenguaje el análisis sintáctico y el análisis semántico estaban unidos para que, al recibir *tokens* no fuese necesario crear un árbol y que el análisis semántico lo recorriese para crear una definición dirigida por la sintáxis. En esta asignatura

vamos a ver que es posible unir el generador de código intermedio a estos dos analizadores en un único gran bloque que recibe *tokens* a la entrada y devuelve código intermedio. Esto se denomina Traducción dirigida por la sintáxis.

Generador de código intermedio

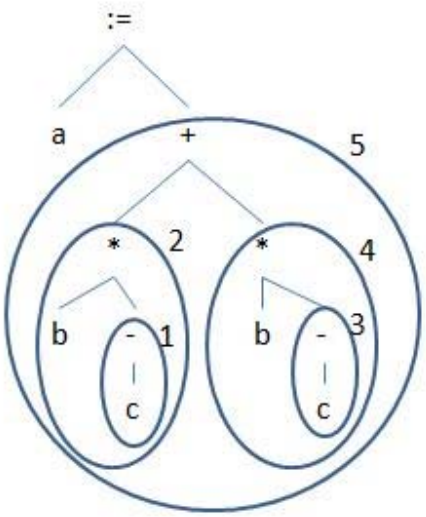
Lenguajes intermedios

El código intermedio podría ser de diferentes formas:

- Representación gráfica:
  - Árboles.
  - Grafos dirigido acíclico.
- Notación Polaca Inversa (RPN).
- Código de 3 direcciones .

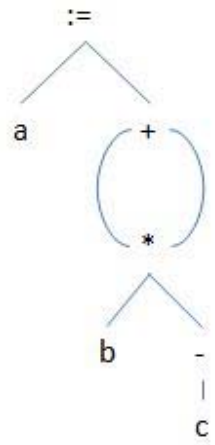
Como ejemplo utilizaremos la siguiente frase de la que suponemos el análisis correcto:
$a := b * - c + b * - c$

Árboles



Como vemos es importante conocer el orden de los operadores.
Podemos utilizar un grafo dirigido acíclico para eliminar redundancia.

GDA



Notación polaca inversa

RPN son sus siglas en inglés. Es una notación donde primero se colocan los operadores y después el código de operación que utiliza esos operadores. Por ejemplo:

$a+b \rightarrow ab+$

$a*b \rightarrow ab*$

$a*b+c \rightarrow ab*c+$

$a+b*c \rightarrow abc*+$

En nuestro ejemplo

$a := b *- c + b *- c \rightarrow abc -* bc -*+:=$ En este caso - es un operador unario, se sabe porque el código es diferente.

De forma conceptual podemos ver la RPN como una linealización de un árbol.

Esta representación parece ideal pero solo con expresiones (sobre todo aritméticas) pero se torna ineficiente con cierto tipo de sentencias (if-then-else, do-while,...), muy comunes en un lenguaje de programación. Por ejemplo, en un if-then-else evalua la rama del Then y del Else antes de evaluar la sentencia if.

$if\ x > 7\ then\ y := 6\ else\ y := 8 \rightarrow x7 > y6 := y8 := IF$ Siendo IF el código de operación de la sentencia if-then-else, considerada en este caso una sentencia triaria.

Esta ineficiencia solo puede resolverse con el método que vamos a utilizar para el código intermedio: el código de 3 direcciones.

Código de 3 direcciones

Son instrucciones como máximo de tres direcciones (operador operando operando). Por ejemplo:

- $x \coloneqq y \; op \; z$
- $x \coloneqq op \; y$

En el ejemplo

$a \coloneqq b * - c + b * - c$

$t_1 \coloneqq \; - c$

$t_2 \coloneqq b * t_1$

$t_3 \coloneqq \; - c$

$t_4 \coloneqq \; b * t_3$

$t_5 \coloneqq \; t_2 + t_4$

$a \coloneqq \; t_5$

Como se ve, cada operación se hace a parte y se van uniendo, siempre en operaciones de 3 direcciones como máximo.

Ejercicio

Diseñar el Generador de Código Intermedio mediante Definición Dirigida por la Sintaxis (DDS) para esta gramática y que obtenga como salida el código de 3 direcciones:

1. S → id := E
2. E → E + E
3. E → E * E
4. E → -E
5. E → (E)
6. E → id

Utilizar los atributos:

- .lugar → Representa la posición [dirección de memoria o registro de la máquina] en la que está ese id (o resultado) en tiempo de ejecución.
- .cod [.código] → Contiene el conjunto de instrucciones en código de 3 direcciones correspondiente a la traducción de ese símbolo gramatical.

Cómo ejemplo:

1. S.cod := … (Únicamente ya que no tiene S.lugar)
2. E.lugar := nuevoTemp(...
   E.cod := E_1.cod || E_2.cod || gen (... '+'...)
6. E.lugar := BuscaLugarTS(id.entrada)